EECS 280 - Lecture 1

Introduction

and Machine Model

eecs280.org

Welcome!

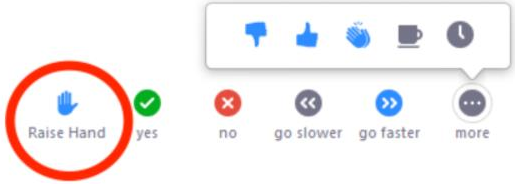
- Feel free to unmute you mic and say hi!
  - Or through the chat feature

- We'll start a few minutes late today to give people time to login

- This slide is just to remind Jon to record the lectures

Using Zoom

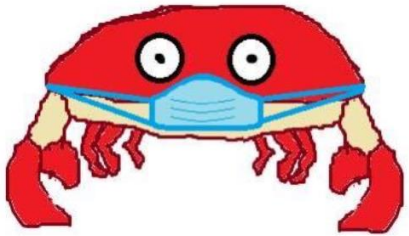
- Have a question / comment?
- Best method is to click "Raise Hand" in Zoom
- I see an ordered list of everyone with their hand raised



- You can also try typing it in chat, but I may miss it

280 Logistics

- We plan to be in-person whenever safe
- Keep an eye on your email regarding next week's lectures

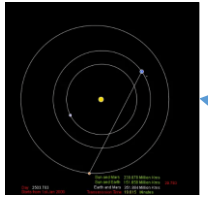


Agenda

- **Introductions**
- Class orientation
- 1st topic of semester: Machine Model

About Me

- Bachelors (CE), Masters (CSE), and PhD (CSE) all from Michigan!
- Research in computer architecture
  - How to make devices like Graphics Processing Units (GPUs) better at non-graphics stuff?



About Me

- When not doing school stuff, you can find me:
  - Triathlon training
  - Cooking
  - Learning about retro games / emulators
- Call me:
  - Jon
  - Dr / Prof Beaumont
  - “Yo Teach!”

<u>Poll:</u> Who are you?

Agenda

- Introductions
- **Class orientation**
- 1st topic of semester: Machine Model

What is EECS 280 about?

- "Computer science is no more about computers than astronomy is about telescopes."

  -Edsger Dijkstra… except not really

- **Generalizable** CS concepts
  - EECS 280 in one word:
    - Abstraction!

No, not that kind of abstract!

Bold and Brash, S. Tentacles

Why Abstraction?

- **Abstraction**: *Removing certain details to focus attention on other, more important details*
- Imagine this world:

"Wanna drive a drive a car? No problem!

If it's a 2013 Ford Taurus, inject a 14.7:1 fuel-air ratio into ignition chambers at 17 PSI every 7.5 ms, igniting via a 23kV spark. To turn left, rotate pinion gear at 18:1 ratio…

And if it's a 2004 Buick LaSabre, inject 16.2:1…"

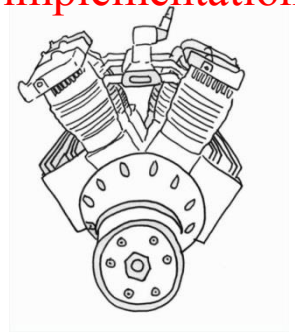
BTW, this is gibberish. I know nothing about cars

- **No thank you!**

Why Abstraction?

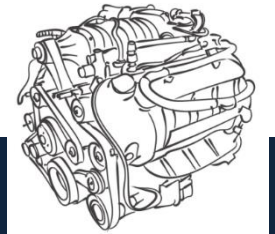
Simple Abstraction

Complex implementation



Rotate wheel clockwise to turn right, push right pedal to accelerate…

Alternate implementation



Why Abstraction?

- Example: In C++, vectors are like fancy arrays
- You can access elements using "[]", but you can also make larger by pushing in the back

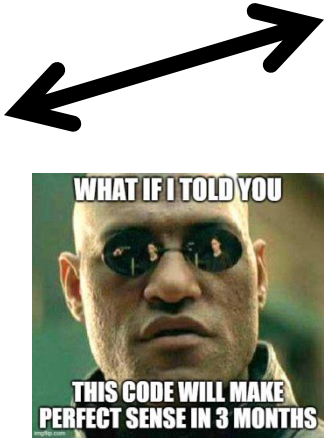
```
vector<int> v = {1, 2}; // v has 2 elements
cout << v[0];           // prints "1"
v.push_back(3);         // now v has 3 elements!
```

- Under the hood, vectors are build using arrays… but it's complicated

Why Abstraction?

Simple Abstraction

```
vector<int> v;
v.push_back(3);
```



Complex implementation

```
template<class T>
void Vector<T>::push_back(const T & v) {
  if (size >= capacity) {
    capacity = 1 << log;
    log++;
    T * newBuffer = new T[capacity];

    for (unsigned int i = 0; i < size; i++)
      newBuffer[i] = buffer[i];
    delete[] buffer;
    buffer = newBuffer;
  }
  buffer [size++] = v;
}
```

Why abstraction?

- Using abstraction in design…
  - Makes it easer to convey how something works
  - Minimizes how much a small change impacts the rest of a complex system
- This will lead us into discussions of Abstract Data Types (classes), polymorphism, dynamic resource management, recursion….
  - See our syllabus for a complete list of topics!

Resources

All linked from eecs280.org

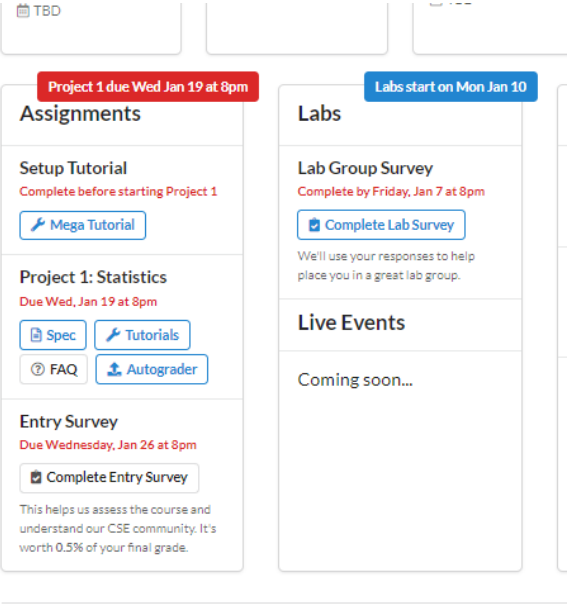
Course Notes	Textbook
Canvas	Announcements and Grades
Piazza	Lecture, Lab, Project Q/A
Gradescope	Turn in Lab Worksheets
autograder.io	Turn in Projects
eecsoh.org	Sign up for office hours
Discord	Informal Course Community

Where to start?

- eecs280.org
  Project 1 Setup Tutorial

- Walks you through setting up your computer for C++ development!

- If you get stuck, that's ok! We're glad to help in office hours.



Lectures

Jon Beaumont

Lecture

Wed 1/5 on Zoom

M & W 12-1:30pm
220 Chrysler

M & W 3-4:30pm
1013 DOW

Office Hours

TBD

Ashley Carroll

Lecture

Wed 1/5 on Zoom

M & W 4-5:30pm
AUD 4 MLB

Office Hours

TBD

James Juett

Lecture

Wed 1/5 on Zoom

M & W 4:30-6pm
Stamps Auditorium

Async Lectures

Office Hours

TBD

Sofia Saleem

Lecture

Wed 1/5 on Zoom

M & W 9-10:30am
Stamps Auditorium

Async Lectures

Office Hours

Remote OH

Tu & Th 10:30-11:30 am

Labs

Labs start next week, Mon Jan 10

- **Attendance at lab section is required for credit.**
  - **Everyone will get credit for first lab**
- You **MUST** attend the lab you are officially scheduled for
  - Indicate in Lab Survey if you have a conflict
- Meet with lab for a bit, then split into groups of 4 to complete a worksheet of exercises
- 1-2 staff will be available for you to ask questions

Lab Group Survey

- We try to match students up to make cohesive and effective lab groups
- Please fill out the lab group survey by this **Friday, Jan 7 at 8pm**
- **Double check** you can attend your registered section!
  - If you're on the waitlist or have a scheduling conflict, let us know on the form and we'll help get it sorted out.

Office Hours

- Check eecs280.org for calendar with full office hour schedule
  - Most will be offered virtually
- GSI/IA OH will start later this week
- My OH: TBA

Exams

- Midterm Exam
  Wednesday, Feb 23 7-9 pm Eastern
- Final Exam
  Tue, April 19 10:30am-12:30 pm Eastern
- Exams will be administered remotely and electronically
- Alternate request forms listed under "Administrative Request Forms" on front page

Grades

- What do I need to do to pass?
  - 70% weighted exam average AND
  - 60% weighted project average

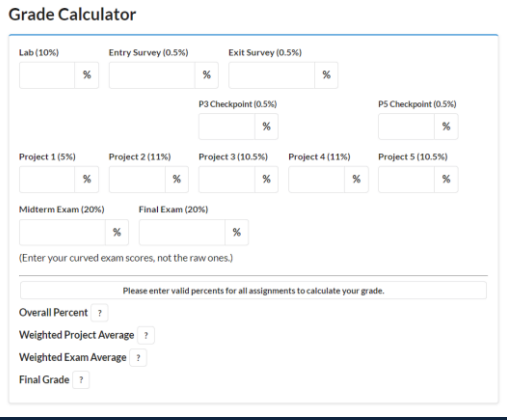
- In the past, the median student has received a grade in the B range

Labs	10%
Projects	49%
Midterm	20%
Final	20%
Computing CARES entry and exit survey	1%

Grades

- Your exam scores may be adjusted higher to account for difficulty of exam
  - We target a mean in the mid-80s
  - Never adjusted downwards
- See syllabus for grading thresholds

Calculate your grade on the website!



Overall %	Grade
0 - 50%	E
50 - 60%	D
60 - 70%	C-
70 - 77%	C
77 - 80%	C+
80 - 83%	B-
83 - 87%	B
87 - 90%	B+
90 - 93%	A-
93 - 97%	A
97 - 100%	A+

Projects

- Read and understand a problem specification.
- Design a solution.
- Implement the solution elegantly and using good coding practices.
- Test and debug your implementation.
- The autograder often slows down near the deadline

Project	Topic	Grade
1	Stats calculator	5%
2	Image processing	11%
3	Euchre simulator	11%
4	Web server	11%
5	Machine Learning	11%

Project grading

- Autograder
  - Grades for correctness and style
  - We provide a very minimal set of public test cases. This is only a small part of what we grade for your full submission.
  - 3 submissions per day
  - We grade the best submission
    - INCLUDING the hidden test cases
  - Linked from eecs280.org

Late Policy

- **<u>No late days</u>**

- Extensions only granted for extreme cases
  - Such as a **medical or personal emergency**
- If something comes up, submit form listed on front page
  - **Needs to be submitted at least 24 hours before deadline if you are able**

Collaboration

Task	Collaboration
Project 1	Individual
Labs	Groups of 4
Projects 2 – 5	One Partner allowed
Exams	Individual

Partnership Traps

- Do not form a partnership without planning how and when you will work on the project together.
- Do not split the work in half. This means you miss out on half the learning from the project. This hurts on exams.
- **Partnerships are optional –** extensions are not granted for a partner not doing work, communicating, etc

Academic integrity

- We want you to collaborate as you learn!
  - But **DON'T SHARE CODE**

DO	DO NOT
Share high level strategies	Share code
Help someone debug	Debug for someone
Explain compiler errors to someone	Fix someone's compiler error
Discuss test strategies	Share test cases

Cheat Checking

- Automated software and manual inspection
- We've reported close to 10% of class in previous semesters
- We only report clear cases of academic dishonesty
- You will not be reported for:
  - Using starter code provided by course instructions
  - Having the same idea as someone else
  - Receiving similar help from the same staff member in OH

Coached Teams

See eecs280.org for more info!

- What is coaching?
  - Connect 1-on-1 with an IA or GSI to plan for your 280 experience.
  - **Tips** and **tricks** for getting the most out of 280
  - Strengthen **metacognitive** skills like exam studying, project planning, optimizing your time, etc.
  - Build your **confidence** in 280 and CS in general
- It's an **opportunity** and a **commitment**.
  - If this sounds like a good fit for you, sign up!
  - But we have limited staff resources, so we also expect students who join to follow-through.

Who doesn't love a good coach?

Well-being

- Don’t let your success in the class be at the expense of your health!
- Eat real food
- Get enough sleep
- Come talk to the staff if you’re feeling overwhelmed
- Other resources:
  - https://caps.umich.edu
  - https://uhs.umich.edu/mentalhealthsvcs
  - https://www.uhs.umich.edu/aodresources

COVID-19

- We also understand that the pandemic affects students in significant and varied ways, and we will work to ensure all students are able to participate in the course.
- Please reach out to us to discuss specific challenges or needs you have.

Agenda

- Introductions
- Class orientation
- **1st topic of semester: Machine Model**

Machine Model

- This class doesn't cover how computers work under the hood
  - Take Computer Organization (EECS 370) to learn more about that
- BUT… it is important to be familiar with a few key ideas about how computers run programs

Program Flow

Compile Time

Runtime

```
[jbbeau@Jon-Desktop scratch]$ g++ hello.cpp -o hello.exe
[jbbeau@Jon-Desktop scratch]$
```

C++ code

Compiler

Executable

```
[jbbeau@Jon-Desktop scratch]$ ./hello.exe
Hello, World!
[jbbeau@Jon-Desktop scratch]$
```

Execute

hello.cpp

```
cout <<
"Hello World!"
<< endl;
```

hello.exe

```
0010001000
```

Compile time terminology

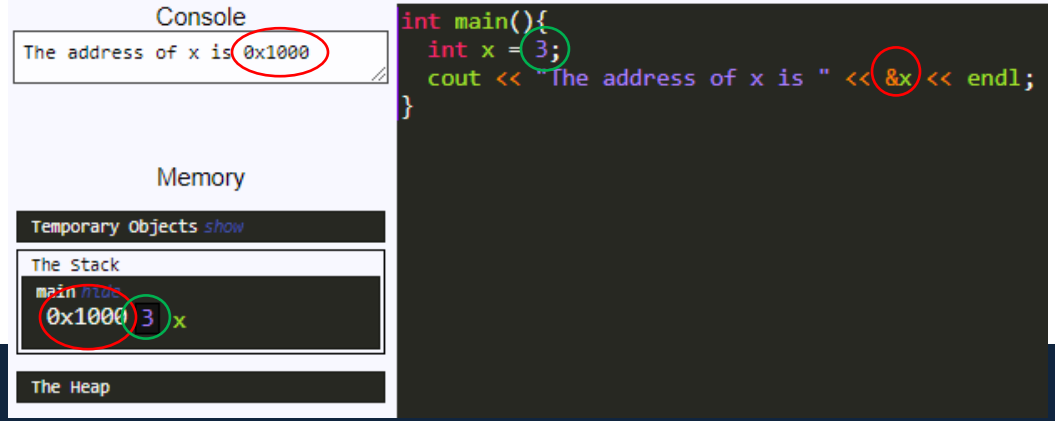
- A **name** is used to refer to some thing.
- A **variable** is a name that refers to an **object** in memory.
- The **scope** of a name is the area of code where it can be used.
- A **declaration** introduces a name and begins its scope.

```
double THRESHOLD = 1000000000;

void helloWorld(double population) {
   if (population < THRESHOLD) cout << "hello small";
   else cout << "hello big" << endl;
}

int main() {
   double p = 7000000000;
   helloWorld(p);
}
```

Poll: Which of these are variables?

Runtime terminology

- An **object** is a piece of data in memory that has value.
- An object lives at an **address** in memory.
  - In C++, you can see what the numerical address is by using the '&' operator



Runtime terminology

- You can use an object during its **lifetime**.
- Lifetimes are managed according to **storage duration**. Three options in C++:
  - **Static**
    Lives for the whole program.
  - **Automatic (AKA Local)**
    Lives during the execution of its local block.
  - **Dynamic**
    You control the lifetime!

Managed by the compiler… we'll learn more about these in lec 14!

Variables != memory objects

- Variables are names that exist at compile time, and can be used within their scope
- Objects are pieces of data that exist during runtime, and can be accessed during their lifetime

Variable

Console

```
The address of x is 0x1000
```

Memory

Temporary Objects show

The Stack

main hide

0x1000 3 x

The Heap

Object

```
int main(){
  int x = 3;
  cout << "The address of x is " << &x << endl;
}
```

Variables != memory objects

- Why distinguish? Isn't there always a 1-to-1 correspondence?
- No, we'll see many counter-examples:
  - Reference variables (many variable names for 1 object)
    - More in lecturer 2
  - Arrays (1 variable name for many objects)
    - More in lecture 4
  - Dynamic memory (0 variable names for 1 object)
    - More in lecture 13

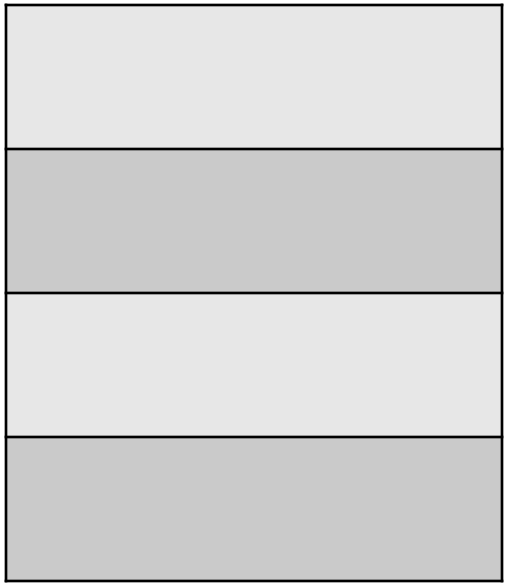
```
int& ref = y;

int array[5];

new int();
```

Value semantics

- Operations in C++ work with **values** of objects in memory
- "Setting equal to" means **copying** the **value**
  - Initialization, function parameters, assignment
- Variables specify which object

```
int x = 42; // initialize value of x to 42
int y = 99; // initialize value of y to 99
x = y;      // copy y's value (99) into x
```

Memory diagram

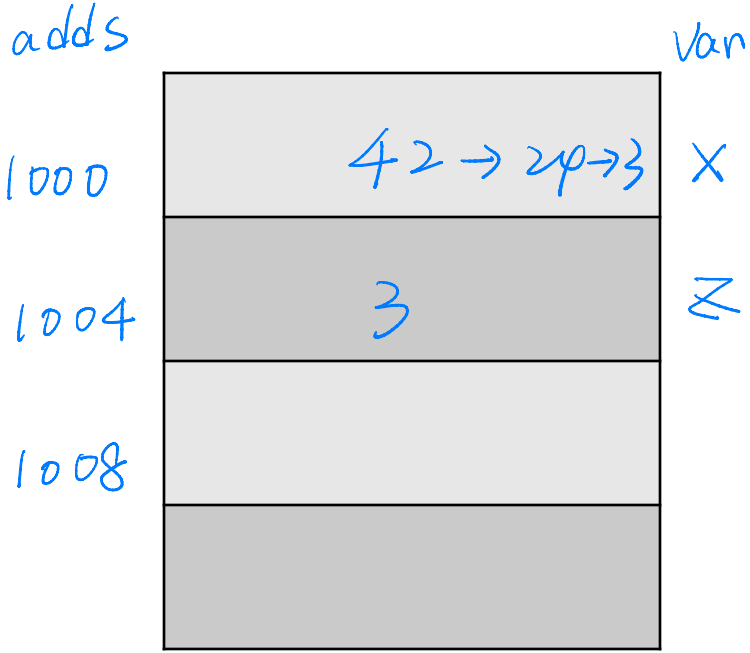


Reference semantics

- With reference semantics, variables are aliases
- In C++, declare a **reference** using the & qualifier
- Declaring a reference does ***not*** create a new object
- Instead, it binds a **new name** to an existing object!

```
int x = 42;    //initialize value of x to 42
int z = 3;     //initialize value of z to 3
int& y = x;    //y and x are two names for one object
x = 24;        //assigns 24 to object named x, AKA y
y = z;         //Does NOT bind y to different object
               //Value semantics used here.
```

Memory diagram



Reference semantics

- Why the heck would we ever use reference semantics??
  - Just seems like a contrived way to modify values
- They let us do fancier things with **function calls**

Next Time

- Continue discussion on machine model
- Talk about testing, debugging and procedural abstraction
  - A.k.a, how to make sure you get full points on projects
- Lingering questions / feedback? I'll include an anonymous form at the end of every lecture: https://bit.ly/3oXr4Ah

Next Time

- Continue discussion on machine model
- Talk about testing, debugging and procedural abstraction
  - A.k.a, how to make sure you get full points on projects
- Lingering questions / feedback? I'll include an anonymous form at the end of every lecture: https://bit.ly/3oXr4Ah